МІНІСТЕРСТОВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп‘ютерної та програмної інженерії
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
«____» _____ 20___р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СПУПЕНЯ «БАКАЛАВР»

Тема: *„Чат-бот для організації витрат через Monobank”*

Виконавець: *Парфенюк Олександра Віталіївна*

Керівник: **к.т.н., доцент** *Моденов Юрій Борисович*

Нормоконтролер: **ст. викл.** *Шевченко А. Т.*

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 "Інформаційні технології", 122 "Комп'ютерні науки та інформаційні технології", "Інформаційні управляючі системи та технології (за галузями)"

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ (*Савченко А. С.*)
«____» _____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту студентки

Парфенюк Олександри Віталіївни

1. Тема дипломного проекту: «Чат-бот для організації витрат через Monobank» затверджена наказом ректора від «22» квітня 2021 р. № 636/ст.

2. Термін виконання роботи: з 10.05.2021 до 14.06.2021.

3. Вихідні дані до роботи: діючі системи організації бюджету, технічна документація програмних засобів, літературні джерела з досліджуваної проблеми.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, розгляд завдання створення чат-бота, дослідження технологій та засобів, розробка програмного продукту, тестування роботи розробленого проекту, висновки.

5. Перелік обов'язкового графічного матеріалу: інтерфейс програм фінансового обліку, структура бази даних, скріншоти роботи розробленого продукту.

КАЛЕНДАРНИЙ ПЛАН

	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Аналіз літератури та джерел за темою дипломного проекту.	10.05.2021р. – 12.05.2021р.	
2	Розробка та затвердження плану дипломного проекту.	13.05.2021р.	
3	Проведення консультації з науковим керівником щодо створення першого розділ.	14.05.2021р.	
4	Аналітичний огляд і постановка задачі.	15.05.2021р. – 18.05.2021р.	
5	Порівняльний аналіз існуючих програм для ведення бюджету.	19.05.2021р. – 22.05.2021р.	
6	Тестування варіантів заміни існуючих програм.	23.06.2021р. – 27.05.2021р.	
6	Створення чат-боту на базі API Monobank та API Telegram.	28.05.2021р. – 04.06.2021р.	
7	Висновки та оформлення пояснювальної записки дипломного проекту.	05.06.2021р. – 08.06.2021р.	
8	Підписання необхідних документів у встановленому порядку.	09.06.2021р. – 10.06.2021р.	
9	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	11.06.2021р. – 12.06.2021р.	

Студентка **(*Парфенюк О.В.*)**

Керівник дипломної роботи (***Моденов Ю.Б.)***

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Чат-бот для організації витрат через Monobank» містить: 59 сторінок, 32 рисунків, 2 таблиці, 11 літературних джерел.

Об’єкт дослідження: оптимізація організації бюджету.

Предмет дослідження: продукт для планування бюджету за допомогою лімітів.

Мета роботи: створення чат-боту з функціями автоматичного введення витрат, планування бюджету за допомогою лімітів, перегляду статистики та одночасного користування банківською карткою двома людьми .

Методи дослідження, технічні та програмні засоби: розробка програмних бібліотек, порівняльний аналіз, обробка літературних джерел.

Отримані результати: створено продукт з усіма зазначеними функціями. Проаналізовано інструменти створення чат-боту для Telegram. Порівняно додатки зі схожим функціоналом.

ЧАТ-БОТ, МОБІЛЬНИЙ БАНКІНГ, ЛІМІТИ, СІМЕЙНИЙ БЮДЖЕТ, TELEGRAM, MONOBANK.

ЗМІСТ

ВСТУП

Програми для обліку фінансів дозволяють не тільки отримувати детальну статистику по всіх операціях, а й позбутися від необхідності зберігати непотрібну інформацію.

Мобільні додатки для обліку фінансових операцій значно зручніше десктопного програмного забезпечення, не кажучи вже про зберігання інформації в Excel або паперовому записнику. Тут можна не тільки швидко вводити дані, а й знаходити інформацію про завершені операції, вносити зміни, експортувати базу даних або відновлювати її з резервної копії.

Проте десктопні програми все ще набагато більш функціональні, і вони не підуть з ринку, так як необхідні для серйозних процесів на кшталт контролю витрат і доходів цілих організацій. Однак для обліку особистих фінансів сучасні мобільні додатки пристосовані набагато краще. При дуже простому і зручному управлінні операціями вони містять всі необхідні для цього засоби, а сам пристрій, на відміну від комп'ютера або ноутбука, завжди під рукою.

У більшості мобільних додатків для контролю фінансів можна створювати кілька рахунків різного типу - кредитні або дебетові картки, вклади, різні активи і так далі. При створенні рахунку вказується назва і початковий баланс. Зазвичай підтримуються різні валюти - рублі, долари, євро, а при внесенні даних про операції по їх конвертації курс може встановлюватися вручну.

Деякі менеджери фінансів мають веб-версію і засоби для експорту в XLS, CSV та інші формати. Навіть при відсутності власного сервісу в інтернеті подібні програми інтегровані з Dropbox і Google Drive, так що резервні копії і таблиці можна зберігати в хмарі.

При розробці програми для обліку фінансів варто брати до уваги не тільки зручність і функціональність, а й безпеку. Метою роботи є розробка чат-бота в Telegram для обліку витрат і планування бюджету.

РОЗДІЛ 1
ОГЛЯД ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ ВИТРАТАМИ

1.1. Аналіз роботи мобільного банкінгу

Основна перевага Інтернет-банкінгу полягає в зручності, адже дистанційно керувати своїми рахунками клієнт може зі звичайного комп'ютера або ноутбука з виходом у всесвітню мережу, не обмежуючи себе ні територіально, ні в часі, економлячи при цьому гроші і нерви.

Основні сервіси, які зараз доступні клієнтам в режимі он-лайн, наступні:

- інформаційний сервіс - надання інформації та виписок за поточними та картковими рахунками, розсилка повідомлень про проведені транзакції, стрічка новин;

- платіжний сервіс - здійснення регулярних і довільних платежів (починаючи від переказів між своїми рахунками, і закінчуючи платежами за ЖКГ, стільниковий і віртуальну зв'язок і т.п.);

- сервіс по роботі з поточними, депозитними та картковими рахунками;

- сервіс по роботі з кредитними продуктами банку (в найближчому майбутньому передбачається отримання он-лайн-кредитів, відкриття / закриття депозитів, і т.п.).

Переваги дистанційного банківського обслуговування:

- економія коштів: програма дозволяє знизити річні витрати з обслуговування рахунків в півтора і більше разів;

- економія часу: менше візитів в банк, і точки самообслуговування;

- ефективність: менше паперової роботи, оперативне виправлення помилок в

платіжних дорученнях;

- оперативність передачі даних: відправка платіжних документів протягом декількох хвилин;

- автоматична перевірка реквізитів документів (номери рахунків, ІПН / КПП одержувача, реквізити банків одержувача і багато іншого);

- застосування шаблонів: збереження бланків для здійснення розрахунків в рублях і іноземній валюті;

- отримання оперативної інформації про курси валют, нові послуги банку, зміну тарифів і умов обслуговування;

- постійний контроль стану рахунку;

- зручний і зрозумілий інтерфейс;

- доступність і простота доступу.

Поряд з численними перевагами використання дистанційного банківського обслуговування не позбавлене і недоліків.

Основним фактором і проблемою, яка стримує розвиток дистанційного банківського обслуговування, є низька кваліфікація клієнтів і брак довіри з боку клієнтів до дистанційних операцій. Найсерйознішою перешкодою є брак довіри з боку клієнтів до дистанційних операцій. Особливо якщо врахувати той факт, що останнім часом почастішали спроби неправомірного отримання персональної інформації користувачів систем дистанційного банківського обслуговування і одне з головних умов для того, щоб клієнт почав користуватися новими можливостями - це забезпечення клієнту максимальної зручності при оплаті за допомогою дистанційного банківського обслуговування.

Наступним фактором, що стримує розвиток дистанційного банківського обслуговування, є клієнтські ризики. Причому при дистанційному банківському обслуговуванні мають місце в основному ризики, пов'язані саме з тим, що банк не передбачив ефективної технології захисту інформації. Йдеться про слабку захищеність інтернету від несанкціонованого доступу. Незважаючи на прагнення розробників інтернет-рішень створювати і вдосконалювати систему захисту переданих повідомлень, численні потенційні небезпеки продовжують з'являтися.

Причини: недоліки операційних систем, програм комунікації та браузерів, людський фактор. Підтримка рівня захисту на належному рівні вимагає значних матеріальних витрат, які можуть собі дозволити в основному великі банки, які розраховують на значні доходи від надання подібних послуг.

Але, все-таки, недоліки, що існують при використанні того чи іншого виду дистанційного банківського обслуговування, переборні в тій чи іншій мірі різними організаційними і технічними способами.

Проте, дистанційне банківське обслуговування є перспективним і значущим напрямком. Так як в даний час російський ринок дистанційного банківського обслуговування розвивається стрімкими темпами, і за останні роки в цій галузі стався якісний зсув.

1.2. Варіантний аналіз особливостей розробки

Платформи для створення мобільних додатків розрізняються між собою набором функцій, цінами, а також тим, як з їх допомогою можна зробити додаток. За останньою ознакою вони діляться на дві основні категорії:

Генератори. Це платформи, які створюють мобільний додаток на основі існуючої веб-сторінки. Генератору вказуємо URL сайту, і він автоматично створює мобільний додаток з тими ж розділами і контентом, що і на сайті.

Конструктори. Це платформи, які дозволяють зібрати додаток самостійно з готових елементів, а контент для нього створять майбутні користувачі. Конструктори містять готові шаблони і елементи інтерфейсу, а також фрагменти функціональності, наприклад, геопозиціонування, відправка повідомлень, робота з банківськими картами і багато іншого.

Є два типи програм, які вміють створювати ці платформи:

Гібридні (PWA). Це, фактично, додатки під веб, адаптовані під екран мобільного пристрою. Вони відкриваються на смартфоні за допомогою браузера.

Нативні. Це, власне, додатки, які встановлюються в операційну систему мобільного пристрою. Нативні додатки найбільш зручні для користувача і вигідні для підприємця.

Створення програми саме по собі може бути безкоштовним, потім є два шляхи. По-перше, можна купити у сервісу його вихідні коди і самостійно підтримувати їх і поширювати додаток. Крім того, можна купити платну підписку, і тоді команда сайту сама опублікує додаток в App Store / Google Play і буде підтримувати його.

Крім плати за підтримку, також доведеться купити аккаунт в App Store або Google Play, який коштує $ 99 і $ 25 відповідно. Щоб окупити витрати, у багатьох платформ є програми лояльності, які дозволяють не тільки зробити додаток, а й заробляти на ньому - наприклад, підключивши рекламу.

На даний час є велика кількість генераторів мобільних додатків, що різняться між собою функціональністю, можливістю, інструментами, вбудованими опціями. Розглянемо детальніше найбільш використовувані.

Appy Pie найдинамічніша платформа для створення додатків в світі. Ця платформа особливо корисна для тих, хто вперше береться за додатки та є новачком в цій галузі. Одна з причин, чому ця платформа так швидко завоювала популярність, полягає у великій кількості пропонованих унікальних функцій. Наприклад, за допомогою Appy Pie можна додати в додаток вбудовані покупки, рекламу, завантажити електронні книги або інший контент, підключити бази даних, інтегрувати соціальні мережі, створити додаток для обміну миттєвими повідомленнями і так далі.

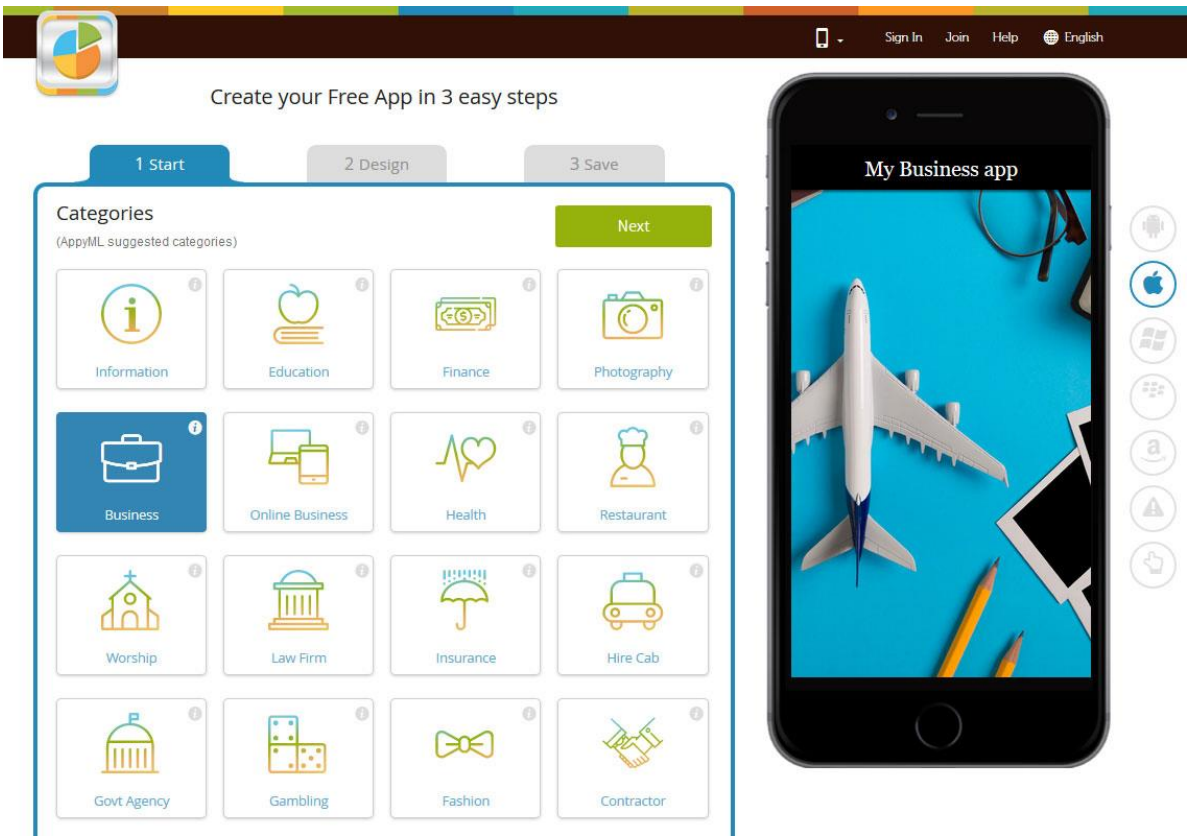


Рисунок 1.1. Платформа Appy Pie

Shoutem - один з кращих продуктів на ринку, і він постійно зростає з моменту відкриття в 2011 році. У своїй останній версії V5 оновлено платформу, значно поліпшено призначений для користувача інтерфейс. Платформа містить шаблони з великою кількістю варіантів настройки і кожен додаток може отримати унікальний зовнішній вигляд і дизайн. Додатки, зроблені в цьому конструкторі, будуть не тільки красивими, але і функціональними. Ця платформа для створення додатків особливо хороша для додатків, пов'язаних з заходами, а крім того відмінно підходить для спільнот, так як завдяки функції соціальної стіни (Social Wall) користувачі можуть ділитися коментарями та фотографіями.

Shoutem робить свій код відкритим і доступним для розробників. Це дозволяє їм залучати більше програмістів для створення додаткових розширень або розробки більшої кількості функцій для конструктора, так як це в цій галузі вони поки відстають. В цілому, дизайн, користувальницький досвід і шаблони Shoutem гарні,

але платформа і раніше обмежена в кількості функцій, пропонованих для створення додатків.

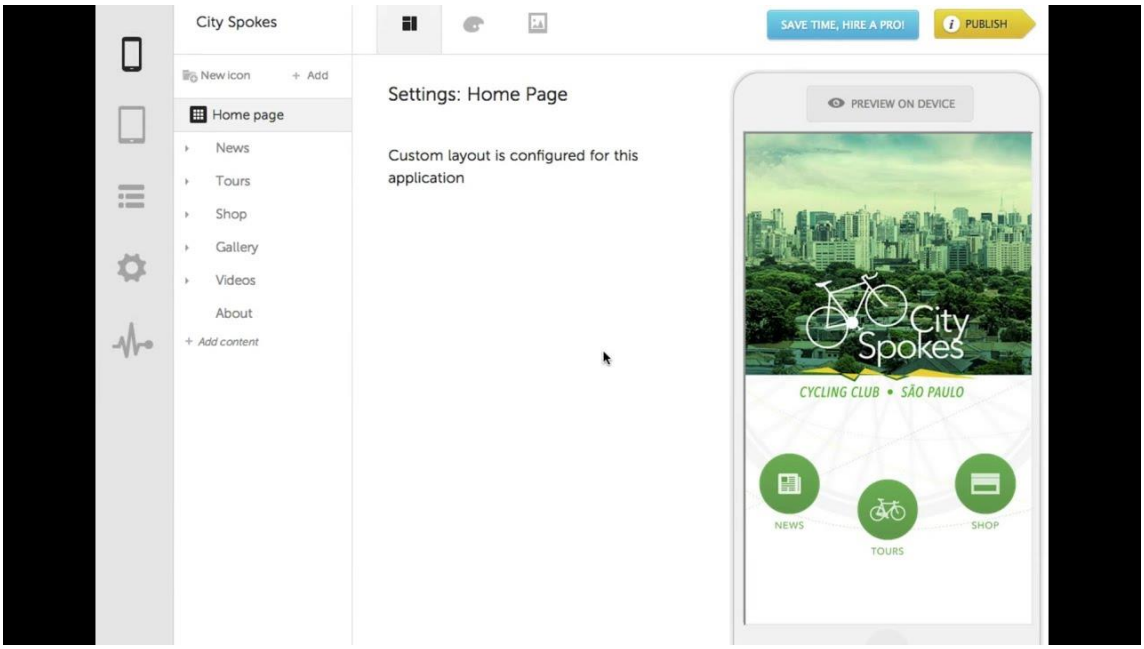


Рисунок 1.2. Платформа Shoutem

Заснований в 2010 році, Swiftic починався як ізраїльський Como, і з тих пір на ньому по всьому світу зроблено більше мільйона додатків.

Компоненти або будівельні блоки, що надаються цією платформою створення додатків, різноманітні і з їх допомогою можна зробити карту лояльності, планувальник зустрічей, e-commerce магазин, можете збирати огляди і оцінки від користувачів або реалізувати додаток для заходу. Більшість додатків, створених на їхній платформі, належать ресторанам, музичним групам і іншим організаціям, які проводять заходи.

Платформа пропонує сім різних шаблонів, які можна комбінувати з шістьма різними стилями навігації. Що робить додаток по-справжньому персональним, так це кольори додатки, фонові зображення і іконки, які можна змінити за своїм бажанням. Редактор конструктора продуманий і простий, а кількість функцій і варіантів дизайну, які вони надають, велика.

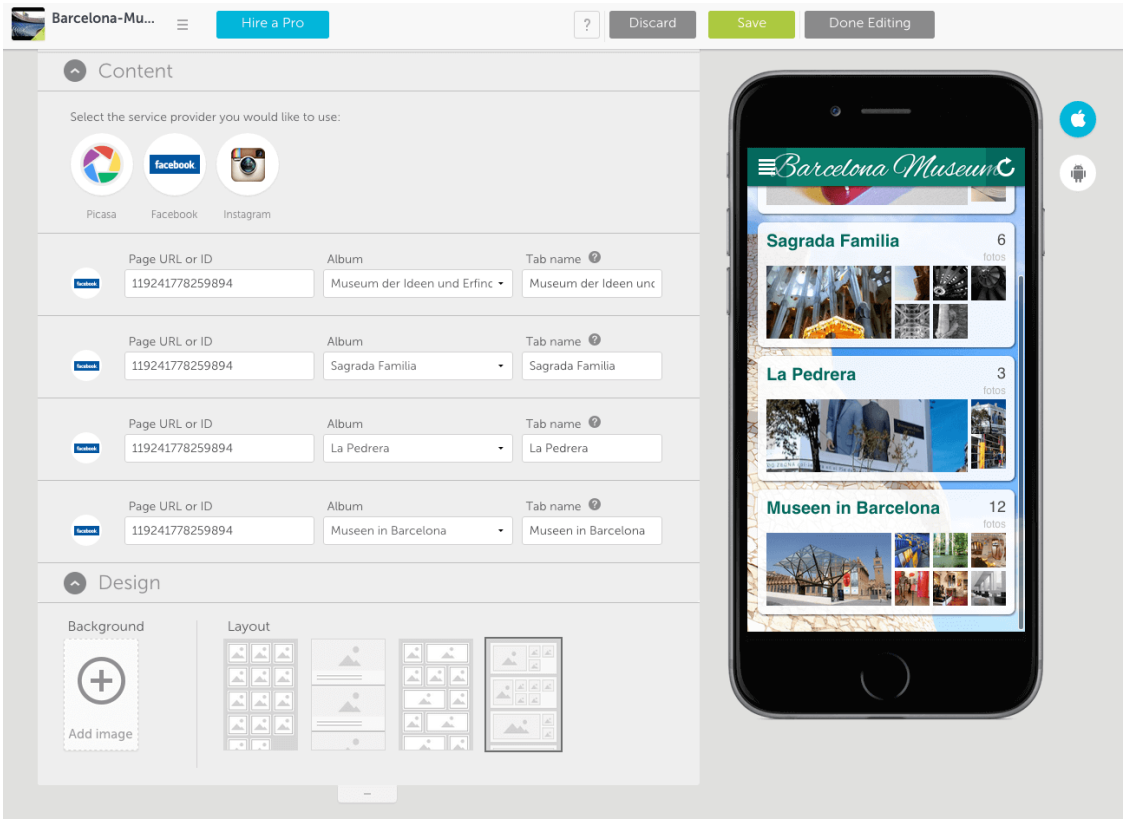


Рисунок 1.3. Платформа Swiftic

Все в цьому конструкторі додатків, починаючи з його назви, вражає. Базована на французькому острові Корсика, платформа для створення додатків пропонує одні з найбільш вражаючих шаблонів. Крім зовнішнього вигляду, конструктор також реалізує деякі самі передові функції - соціальні мережі, чати, геофенсінг, iBeacons і багато іншого.

Платформа з гордістю демонструє додатки, створені на їхній платформі, щоб можна було зрозуміти якість і можливості, які вона пропонує.

Мало того, що їх шаблони красиві, але вони також пропонують досить конкурентоспроможну ціну для нативних додатків. Відмінні додаткові функції, такі як push-повідомлення, чати і т.д дають велику гнучкість при створенні додатків. Однак одним з недоліків є відсутність власного компонента інтернет-магазину, але є можливість інтегрувати сторонні магазини, на кшталт Amazon, Etsy, Shopify і т.д.

Рисунок 1.4. Платформа GoodBarber

BuildFire - одна з найбільш надійних платформ, за допомогою якої вже більше 30 000 компаній створили свої додатки. Більшість їхніх клієнтів - підприємства бренди. BuildFire називає себе однією з провідних платформ на ринку прискореної розробки додатків.

Зручні панелі управління і адміністрування полегшують процес випуску оновлень. Платформа популярна серед клієнтів завдяки простоті використання, можливостям швидкого перенастроювання, а також широким можливостям зміни програми. Можна вносити зміни на льоту і навіть тестувати ці зміни в режимі реального часу.

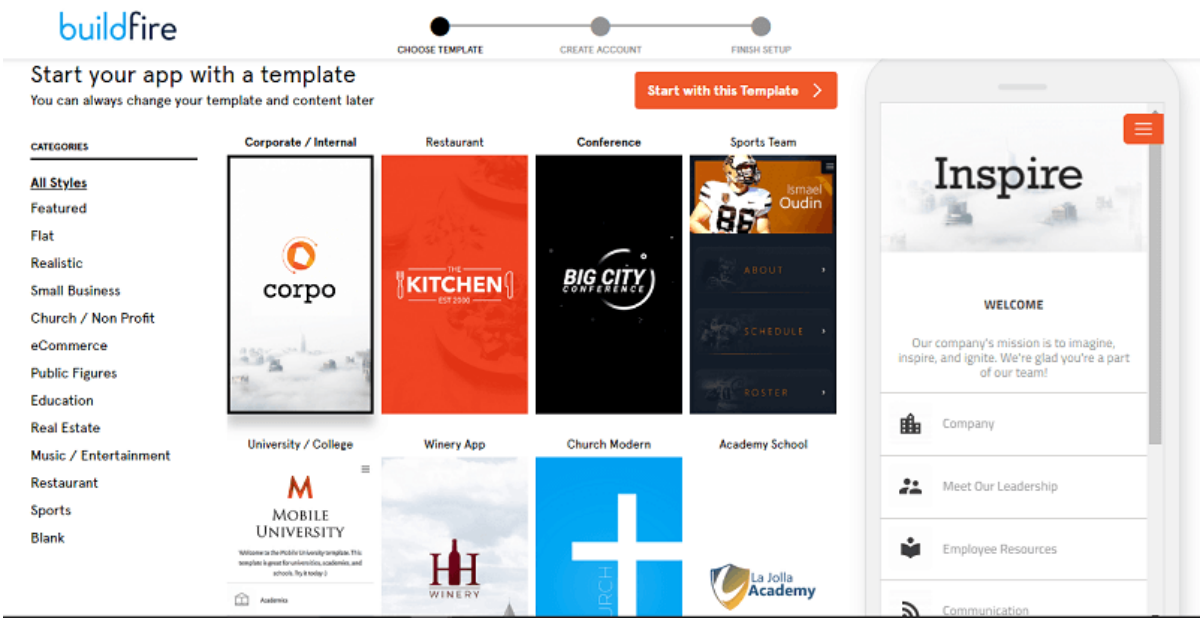


Рисунок 1.5. Платформа BuildFire

Незважаючи на великий перелік можливостей у даних платформ є серйозний недолік, а саме, платна основа. Саме цей фактор робить неможливим використання даних платформ для власних потреб чи в навчальних цілях.

Саме тому актуальною є розробка власного генератора мобільних додатків.

1.3. Огляд аналогічних програм

Monefy - досить популярний фінансовий менеджер, відомий багатьом користувачам. Тут і перегляд витрат «на зрозумілому і інформативному графіку і детальна інформація в списку під ним», і режим бюджету для більш ефективного контролю витрат за певний період. Другий - Money Manager. Програма пропонує програму-компаньйон для ПК, що ставить її в один ряд з Finance PM. Не обійшлося і без управління кредитними і дебетовими картами.

В якості тестового обладнання застосовувався планшет DEXP Ursus 8EV2 3G (Android 4.4.2, процесор MT8382, 4 x 1300 МГц, 1 Гбайт ОЗУ).

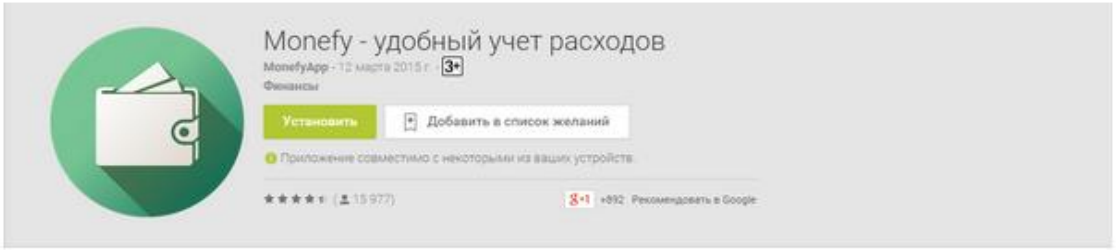


Рис. 1.6. Monefy

Головне, що вміє додаток - це синхронізувати будь-які пристрої через особистий Dropbox-аккаунт. Загалом, бюджет можна редагувати відразу на декількох пристроях, тобто фінансовий менеджер дійсно стає сімейним планувальником.

Основні функції:

- інтуїтивно зрозумілий і зручний інтерфейс,
- миттєве додавання нових записів,
- перегляд витрат на зрозумілому і інформативному графіку і детальна інформація в списку під ним,
- управління категоріями,
- безпечна синхронізація через особистий Dropbox аккаунт,
- вибір звітного періоду,
- режим бюджету для більш ефективного контролю витрат за певний період,
- створення резервних копій та експорт даних в один клік.

Таблиця 1.1

Характеристики додатку Monefy

Інтерфейс	відмінно
Можливість самостійного налаштування	добре
Максимальне навантаження на систему (CPU / RAM)	0.1% / 52.2-62.6 Мбайт
Підтримка декількох валют	є
управління	відмінно

Продовження таблиці 1.1

планувальник	є
кілька гаманців	є
захист паролем	є
Резервне копіювання	є
стабільність	відмінно
Веб-інтерфейс	немає
Робота з хмарою	є

Для початку натиснемо на «+», чи то пак «дохід». Тут вбиваємо суму доходу, вибираємо категорію, наприклад, зарплата і присвоюємо доходу рахунок - готівкові або карта. Вся операція займає мінімум часу, завдяки ну дуже інтуїтивному керуванню.

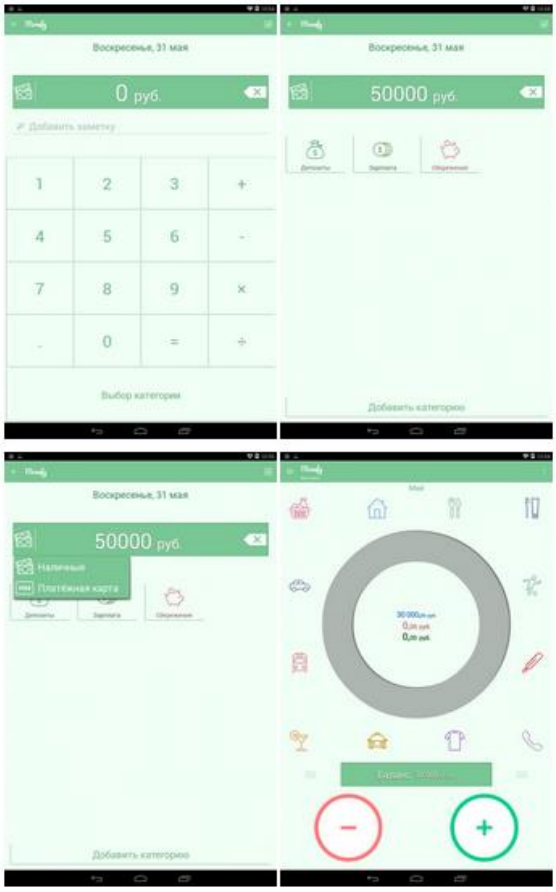


Рис. 1.7. Дохід

З витратами все те ж саме тільки категорій буде побільше, наприклад, гроші можна витратити на зв'язок, розвагу або їжу. Все це справа відразу відображається в круговій діаграмі на головному екрані, а для зручності навколо її можна знайти масу ярликів для швидкого контролю витрат.

Нижче кожного ярлика вказаний відсоток витрат так, що можна побачити на що ви витрачали кошти, а якщо натиснути на будь-яку іконку, то в діаграмі висвітиться сума витрат. Треба відзначити, що процес контролю витрат дійсно не тільки швидкий, але і дуже зрозумілий.

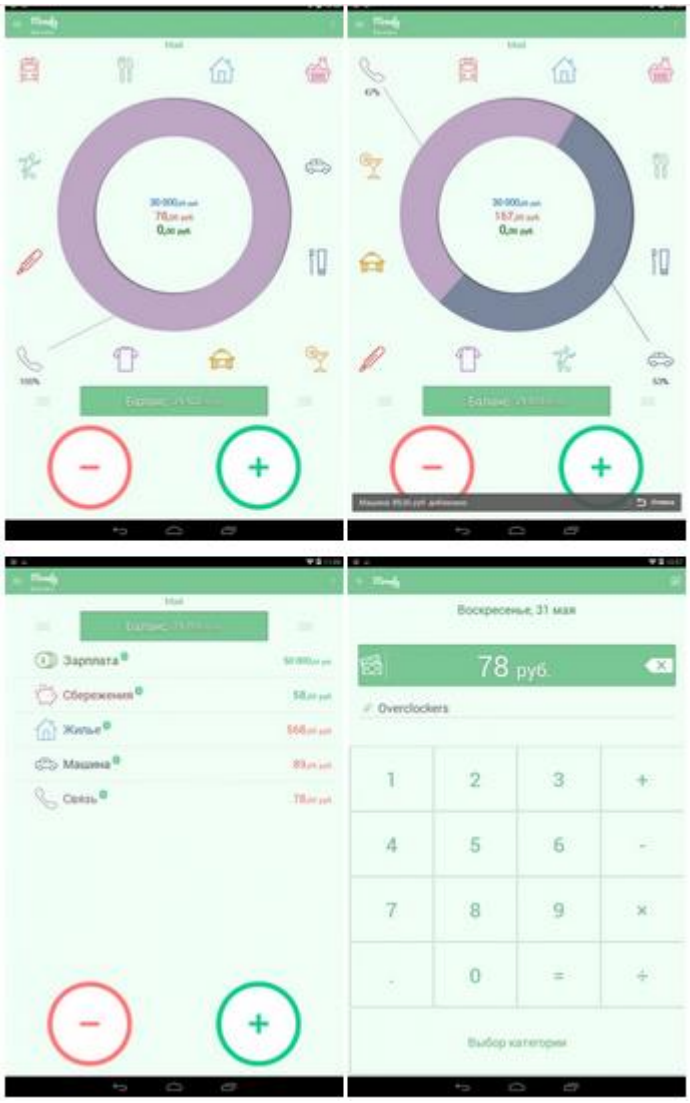


Рис. 1.8. Витрати

Загалом, Monefy можна було б сміливо назвати кращим, якби не ще одна програма попереду.

Домашня бухгалтерія від Keepsoft.

Одна з перших програм такого роду від розробників із СНД, розвиток якої ведеться ще з початку двохтисячних років. За її допомогою можна вести облік доходів і витрат, складати бюджет та будувати довгострокові фінансові плани.

Кількість рахунків, з якими можна працювати, не обмежена. Картка витрат дозволяє розбивати їх за категоріями та підкатегоріями, а також обирати, з якого рахунку буде здійснюватися списання. Таким чином, можна відстежувати доходи і витрати для всіх членів сім'ї.

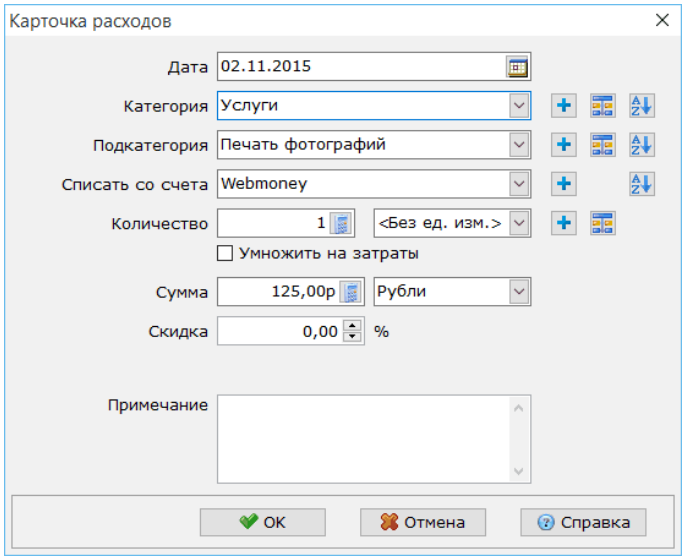


Рис. 1.9. Keepsoft

Одна зі основних можливостей – додавання кредитів і грошей, узятих у борг. Причому програма дозволяє розрахувати графік їхнього повернення, і, якщо потрібно, можна налаштувати регулярні нагадування про це.

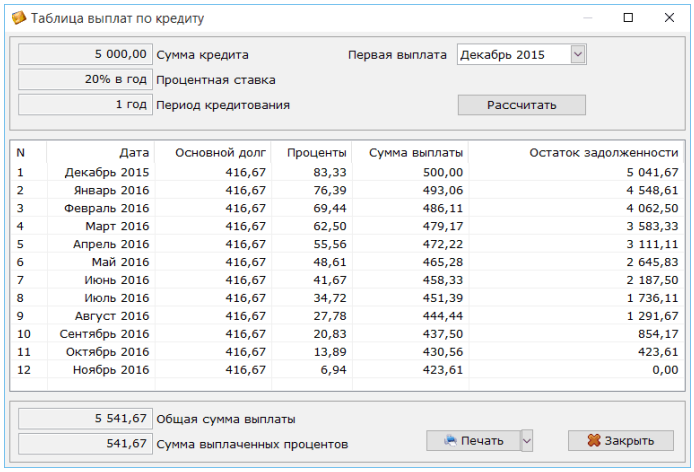


Рис. 1.10. Keepsoft

Наочні звіти і діаграми дозволяють проводити детальний аналіз та робити правильні висновки. Окремо варто відзначити функцію імпорту/експорту як у файли звичного Microsoft Office, так і в інших форматах. Крім того, тут доступні практично всі валюти світу на вибір, що може дуже стати в пригоді деяким користувачам.

Таблиця 1.2

Характеристики додатку Keepsoft

Інтерфейс	задовільно
Можливість самостійного налаштування	добре
Максимальне навантаження на систему (CPU / RAM)	0.1% / 44,8 Мбайт
Підтримка декількох валют	є
управління	відмінно
планувальник	є
кілька гаманців	є
захист паролем	є
Резервне копіювання	є
стабільність	відмінно
Веб-інтерфейс	немає
Робота з хмарою	є

Проаналізувавши вищеописані додатки можна дійти висновку, що в розглянутих системах э певні мінуси, а саме:

- необхідність вручну записувати кожну витрату,
- відсутність інструментів планування витрат,
- вартість продукту,
- наявність непотрібних функцій,
- відсутність одночасного сумісного ведення.

Звичайно, завдяки розвитку мобільного банкінгу та додатку Monobank ми можемо бачити усі наші витрати без необхідності самостійного введення інформації у вигляді графіка за обраний вами період, проте на практиці описаних вище інструментів або недостатньо для планування бюджету, або ж вони є недостатньо оптимізованими.

1.4. Постановка задачі

Постановка задачі дослідження. Спроектувати та розробити систему, що передбачає:

- ведення особистих фінансів,
- розподіл витрат,
- аналіз витрат вказаного періоду,
- одночасне сумісне використання,
- наявність інструменту планування (ліміти).

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ОБРАНИХ МЕТОДІВ ДОСЛІДЖЕННЯ

2.1. Вибір форми реалізації

З появою сімейного бюджету з’явилась потреба у спільному рахунку для здійснення покупок та грошових переказів. Для даних цілей було обрано Monobank, осільки користувач отримує фізичну карту, та «мобільний варіант» який діє на базі NFC. Одна з функцій мобільного додатку Monobank є сповіщення про транзакцію та баланс рахунку. Таким чином, користувач фізичної карти не має швидкого доступу до цієї інформації.

Для вирішення даної проблеми було обрано платформу Telegram, що на сьогодні є найбільш захищеним сервісом для обміну повідомленнями з можливістю створення чат-боту.

Чат-бот - це програма штучного інтелекту, яка може імітувати розмову (або чат) із користувачем природною мовою за допомогою додатків для обміну повідомленнями, веб-сайтів, мобільних додатків або по телефону.

Однією з найбільших переваг чат-ботів є те, що, на відміну від програм, вони не завантажуються, їх не потрібно оновлювати, і вони не займають місця в пам'яті телефону. Інша - це те, що ми можемо мати декількох ботів, інтегрованих в одному чаті. Таким чином ми б уникнули переходу від одного додатка до іншого відповідно до того, що нам потрібно в кожну хвилину.

Залежно від їх використання, чат-боти можуть бути як відкритими, так і закритими. Відкриті чат-боти - це ті, які використовують штучний інтелект для обробки мови та навчання на їх взаємодії з користувачами. Закриті чат-боти - це ті, які лише і виключно виконують розмову або сценарій. Створений чат-бот відноситься саме до цього типу.

З кожним роком зростає популярність мобільних додатків і зростає кількість проектів, але з ростом проектів розширюється спектр завдань і часто закладених спочатку способів для гнучкої розробки нових функцій не вистачає. З огляду на те, що в даному проекті мається на увазі велику кількість модулів і очікується зростання їх кількості в майбутньому, до архітектури в цілому слід підходити так, щоб було легко тестувати окремі модулі, а також щоб у майбутньому через змін не доводилося переписувати все додаток.

Тому було вирішено взяти в основу підхід Роберта Мартіна під назвою Clean Architecture. [24]

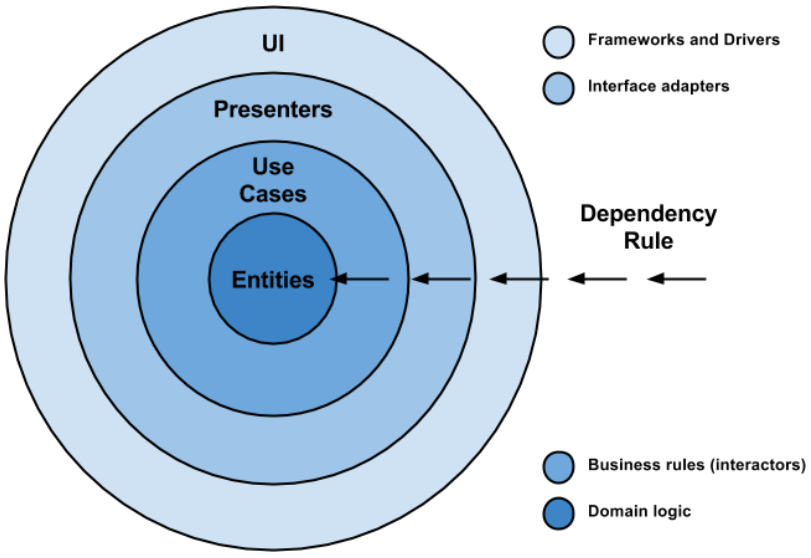


Рисунок 2.1. Схематичне зображення підходу Clean Architecture

Основні принципи Clean Architecture перераховані тут, і пов'язані зі схемою на малюнку 2.1.

Незалежність від структури. Архітектура не залежить від бібліотек або SDK, програма не перебуває в рамках підходів, запропонованих бібліотеками.

Нестування Бізнес-логіку можна протестувати без користувальницького інтерфейсу, бази даних, веб-сервера або будь-яких інших зовнішніх елементів.

Незалежність від призначеного для користувача інтерфейсу. Інтерфейс може легко змінюватися без зміни іншої системи. Наприклад, веб-інтерфейс можна

замінити на призначений для користувача інтерфейс консолі, не змінюючи бізнес-правила.

Незалежність від бази даних. Важливо, щоб можна було легко поміняти Oracle або SQL Server на Mongo, BigTable або CouchDB легко і без зміни бізнес-правил.

Розробляючи додаток так, слід розділити його на 3 основних модуля - Presentation модуль, де буде розташовуватися весь патерн MVP і відображення; Domain модуль, де будуть чисті Java об'єкти; Data модуль, де буде описано методи доступу до даних (з мережі і з збережених на пристрої файлів). Це зображено на рис. 2.2.

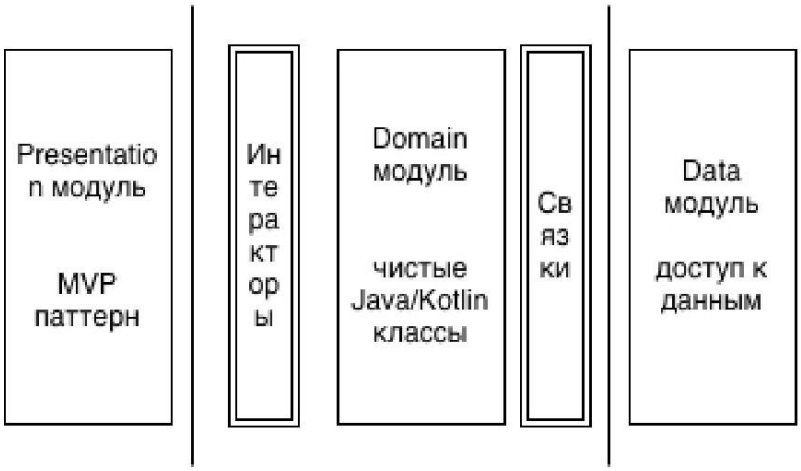


Рисунок 2.2 – Поділ програми на модулі по Clean Architecture

2.2. API як інструмент взаємодії

API - це специфікація можливих взаємодій з компонентом ПО. Наприклад, якби машина була компонентом ПО, в її API містилася б інформація про те, що машина може прискорюватися, гальмувати і включати радіо. Її API також б включало інформацію про те, як прискорюватися. API працює на 3 рівнях:

- додаток - програми, які ви використовуєте на смартфоні, або ПО;
- програмування. Програмісти використовують API, щоб писати код;
- Інтерфейс - як ви взаємодієте з додатком.

Приклад: Уявіть офіціанта в ресторані. Ви - клієнт, вибираєте замовлення в меню. Кухня - виконавець вашого замовлення. Вам потрібен посередник, який повідомить про замовлення на кухню і принесе вам їжу на стіл. Їм не може бути шеф-кухар, тому що він готує на кухні. Вам потрібен хтось, хто пов'яже клієнта і шеф-кухаря. І тут в гру входить офіціант - API. Офіціант приймає ваше замовлення, приносить його на кухню, каже, що ви замовили. Потім доставляє відповідь, або їжу вам. Більш того, якщо все зроблено правильно, ваше замовлення не впаде.

Публічні API випускаються такими компаніями, як Slack і Shopify, в надії на те, що розробники будуть їх використовувати на своїх платформах. Компанії діляться набором вступних параметрів, які розробники використовують, щоб досягти якогось результату. Публічне API можна використовувати без проблем - доступ до документації можна отримати без проблем.

Приватний API використовуються всередині компанії. Якщо у компанії багато програмних продуктів, приватне API використовується, щоб програми розмовляли між собою. Компоненти API можуть змінюватися за бажанням компанії, тоді як зміни в публічному API можуть викликати невдоволення тих, хто їм користується.

У даній роботі використовуються два види API – Telegram та Monobank, які є публічними та застосовуються на рівні інтерфейсу . За допомогою них я маю можливість бути посередником між чат-ботом та банком.

Monobank API має токен користувача, який використовується для ідентифікації та доступу до певної інформації по рахунку. У даній роботі він використовується для того, щоб обрати потрібну карти і отримувати дані про транзакції.

Telegram API використовується для отримання та обробки запитів користувача, інофрмації від банку та від бази даних.

2.3. Аналіз принципу розробки web-сервісів за допомогою фреймворку Express

Express використовується для розробки додатків досить давно і завдяки своїй стабільності міцно займає позицію одного з найпопулярніших фреймворків Node.js.

Для цього фреймворка існує велика кількість докладних інструкції та описів, які складені розробниками, що перевіпили його ефективність на практиці. Тому саме з Express рекомендується починати роботу, якщо ви маєте намір навчитися створювати додатки на платформі Node.js.

Основна особливість цього фреймворка полягає в тому, що для Express характерний невеликий обсяг базового функціоналу. Всі інші необхідні вам функції потрібно буде добирати за рахунок зовнішніх модулів. По суті, Express в чистому вигляді - це сервер і у нього може не бути жодного модуля.

Завдяки такому мінімалізму розробник спочатку отримує в своє розпорядження легкий і швидкий інструмент, який він може розширювати і розвивати.

При цьому важливо, що вибір модулів для Express не пов'язаний ні з якими обмеженнями: ні з кількісними, ні з функціональними.

В результаті, цей фреймворк забезпечує розробнику можливість вирішувати будь-які завдання, не обмежуючи його при цьому у виборі засобів.

З одного боку, не може не радувати той факт, що відсутність готових універсальних рішень фактично означає, що кожне створюване додаток буде унікальним[8].

З іншого боку, розробнику потрібно самостійно відбирати і організовувати модулі, а це передбачає великий обсяг роботи і відповідно, вимагає від розробника більше часу і зусиль.

Ось приклад:

```
var express = require ( 'express');
var app = express.createServer ();
app.get ( '/', function (req, res) {
   res.send ( 'Hello World');
```

```
});
app.listen (3000);
```

Тут створюється web-сервер, «слухає» порт 3000 і обробляє запит до /, відповіддю на який виводиться рядок Hello World.

У express вбудована потужна система маршрутизації. наприклад:

```
app.get ( '/ user /: id', function (req, res) {
   res.send ( 'user' + req.params.id);
});
```

Даний код обробляє запити / user / foo, для яких автоматично виставляється значення foo для змінної req.params.id. Для опису маршрутів ви так само можете використовувати регулярні вирази.

Якщо ви хочете обробляти POST запити, то вашому додатку необхідно використовувати спеціальний middleware - bodyParser. Підключається він дуже легко: app.use (express.bodyParser ()). BodyParser обробляє тіла application / x-www-form-urlencoded і application / json запитів і виставляє для них req.body. Ось приклад:

```
app.use (express.bodyParser ());
app.post ( '/', function (req, res) {
   console.log (req.body.foo);
   res.send ( 'ok');
});
```

В даному прикладі на консоль виводиться значення змінної foo з тіла запиту, а у відповідь на запит повертається рядок ok.

Крім bodyParser є ще кілька middleware'ей:

```
app.use (express.logger (...));
app.use (express.cookieParser (...));
app.use (express.session (...));
app.use (express.static (...));
app.use (express.errorHandler (...));
```

logger відповідає за логування HTTP запитів, cookieParser - за обробку cookies, session - за роботу з сесіями, static - за роботу зі статичним контентом (css, javascript,

картинки), errorHandler - за обробку помилок. Більш докладно про них можна дізнатися з документації.

Крім того, express має підтримку різних шаблонних движків. Наприклад, мови jade (автором так само є TJ). Ось приклад використання jade з express:

```
app.get ( '/', function (req, res) {
   res.render ( 'index.jade', {title: 'My Site'});
});
```

Імена файлів шаблонів мають вигляд <ім'я>. <Движок>, де <движок> - ім'я модуля, який необхідний для обробки тіла шаблону. Наприклад, шаблон layout.ejs «повідомить» express, що необхідно зробити require ('ejs') перед обробкою шаблону. Завантаження модуль повинен експортувати метод exports.compile (str, options) і повертати Function[5].

Основні можливості express:

- гнучка система маршрутизації запитів;
- перенаправлення;
- уточнення контенту;
- особлива увага до продуктивності;
- опрацювання подань і підтримка часткових шаблонів;
- підтримка конфігурацій на основі оточень;
- оповіщення, інтегровані з сесіями;
- максимальне покриття тестами;
- утиліти для швидкої генерації остова додатків.

Крім того:

- підтримка сесій;
- кеш API;
- підтримка mime;
- підтримка ETag;
- постійні оповіщення;
- підтримка кук;
- JSON RPC;

- логування.

2.4. Аналіз особливостей розробки бази даних

Сучасні інформаційні технології побудовані на принципах взаємодії з базами даних. Бази даних грають велику роль у житті більшості людей. Вони поширені всюди, починаючи від невеликих підприємств, які утримують власні бази товарів, працівників, партнерів, постачальників і закінчуючи базами даних масштабу глобального масштабу - держави чи кількох держав, наприклад, реєстраційні дані про людину чи автомобіль. Бази даних дозволяють ефективно розмістити та упорядкувати дані, що дозволить отримати ефективний пошук та взаємодію. Тому вони займають провідне місце на ринку програмних продуктів.

База даних (БД) - сукупність даних, що зберігаються у відповідності зі схемою даних і відображають стан об'єктів та їх взаємозв'язків, маніпулювання якими виконуються відповідно до правил засобів моделювання даних.

Відмінні ознаки:

- БД зберігається і обробляється в обчислювальній системі. Таким чином, будь-які інші сховища інформації (архіви, бібліотеки, картотеки)базами даних не є;
- дані в БД логічно структуровані (систематизовані) з метою забезпечення можливості їх ефективного пошуку та обробки в обчислювальній системі. Структурованість дозволяє явне виділення складових частин (елементів), зв'язків між ними, а також типізацію елементів і зв'язків, при якій з типом елемента (зв'язку) співвідноситься певна семантика і допустимі операції;
- БД включає схему, або метадані, що описують логічну структуру БД у формальному вигляді[4].

Реляційна база – це колекція пов'язаних таблиць. Різниця між базою даних і реляційною базою даних знаходиться в будові таблиць. У реляційній базі даних таблиці побудовані так, що є логічний зв'язок між ними. На підставі інформації, яка є в одній таблиці, можна отримати відповідну інформацію з іншої таблиці.

Існує декілька етапів при розробці баз даних.

Етап концептуального проектування полягає в описі і синтезі інформаційних вимог користувачів у початковий проект БД. Вихідними даними можуть бути сукупність документів користувача при класичному підході або алгоритми додатків при сучасному підході. Результатом цього етапу є високорівневе подання (у вигляді системи таблиць БД) інформаційних вимог користувачів на основі різних підходів.

Спочатку вибирається модель БД. Потім створюється структура БД, яка заповнюється даними за допомогою систем меню, екранних форм або в режимі перегляду таблиць БД. Тут же забезпечується захист і цілісність (у тому числі посилальна) даних за допомогою СУБД або шляхом побудови тригерів.

У процесі логічного проектування використовується високорівневе подання даних у структуру СУБД, що використовується. Основною метою етапу є усунення надмірності даних з використанням спеціальних правил нормалізації. Ціль нормалізації - мінімізувати повторення даних і можливі структурні зміни БД при процедури оновлення. Це досягається розділенням (декомпозицією) однієї таблиці в дві або декілька з подальшим використанням при запитах операції навігації. Зауважимо, що навігаційний пошук знижує швидкодію БД, тобто збільшує час відгуку на запит. Отримана логічна структура БД може бути оцінена кількісно за допомогою різних характеристик (число звернень до логічних записів, обсяг даних в кожному додатку, загальний обсяг даних). На основі цих оцінок логічна структура може бути вдосконалена з метою досягнення більшої ефективності.

Спеціального обговорення заслуговує процедура управління БД. Вона найбільш проста в режимі одного користувача. У багатокористувацькому режимі і в розподілених БД процедура сильно ускладняється. При одночасному доступі декількох користувачів без прийняття спеціальних заходів можливе порушення цілісності бази даних. Для усунення цього явища використовують систему транзакцій і режим блокування таблиць або окремих записів.

Транзакція - процес зміни файлу, запису або бази даних, викликаний передачею одного вхідного повідомлення. Особливості блокування і варіанти блокування далі будуть розглянуті окремо.

На етапі фізичного проектування вирішуються питання, пов'язані з продуктивністю системи, визначаються структури зберігання даних та методи доступу.

Взаємодія між етапами проектування та словникової системою необхідно розглядати окремо. Процедури проектування можуть використовуватися незалежно в разі відсутності словникової системи. Сама словникова система може розглядатися як елемент автоматизації проектування.

Засоби проектування і оціночні критерії використовуються на всіх стадіях розробки. В даний час невизначеність при виборі критеріїв є найбільш слабким місцем у проектуванні БД. Це пов'язано з труднощами опису та ідентифікації великої кількості альтернативних рішень.

Простішою є справа при роботі з кількісними критеріями, до яких відносяться час відповіді на запит, вартість модифікації, вартість пам'яті, час на створення, вартість на реорганізацію. Утруднення може викликати протиріччя критеріїв одне одному.

У той же час існує багато критеріїв оптимальності, які є невимірними властивостями, важко виразність у кількісному поданні або у вигляді цільової функції.

До якісних критеріїв можуть ставитися гнучкість, адаптивність, доступність для нових користувачів, сумісність з іншими системами, можливість конвертації в іншу обчислювальне середовище, можливість відновлення, можливість розподілу і розширення.

Процес проектування є тривалим і трудомістким і зазвичай триває кілька місяців[7].

MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які притаманні об'єктно-реляційних баз даних.

На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати.

Але, навіть враховуючи всі недоліки традиційних баз даних і переваги MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні. В якійсь ситуації MongoDB дійсно поліпшить продуктивність вашої програми, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних - в традиційних БД.

Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Одним з популярних стандартів обміну даними та їх зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних в MongoDB в цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (Бісон) або скорочення від binary JSON.

BSON дозволяє працювати з даними швидше: швидше виконується пошук і обробка. Хоча треба зазначити, що BSON на відміну від зберігання даних в форматі JSON має невеликий недолік: в цілому дані в JSON-форматі займають менше місця, ніж в форматі BSON, з іншого боку, даний недолік з лишком окупається швидкістю.

MongoDB написана на C ++, тому її легко перенести на найрізноманітніші платформи. MongoDB може бути розгорнута на платформах Windows, Linux, MacOS, Solaris. Можна також завантажити вихідний код і самому скомпілювати MongoDB, але рекомендується використовувати бібліотеки з офсайта.

Система зберігання даних в MongoDB представляє набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Всі вторинні вузли зберігають цілісність і автоматично оновлюються разом з оновленням головного вузла. І якщо основний вузол з якихось причин виходить з ладу, то один з вторинних вузлів стає головним.

Відсутність жорсткої схеми бази даних і в зв'язку з цим потреби при щонайменшій зміні концепції зберігання даних перестворювати цю схему значно полегшують роботу з базами даних MongoDB і подальшим їх масштабуванням. Крім того, економиться час розробників. Їм більше не треба думати про перестворюванні бази даних і витрачати час на побудову складних запитів[7].

Однією з проблем при роботі з будь-якими системами баз даних є збереження даних великого розміру. Можна зберігати дані в файлах, використовуючи різні мови програмування. Деякі СУБД пропонують спеціальні типи даних для зберігання бінарних даних в БД (наприклад, BLOB в MySQL).

На відміну від реляційних СУБД MongoDB дозволяє зберігати різні документи з різним набором даних, однак при цьому розмір документа обмежується 16 мб. Але MongoDB пропонує рішення - спеціальну технологію GridFS, яка дозволяє зберігати дані за розміром більше, ніж 16 мб.

Система GridFS складається з двох колекцій. У першій колекції, яка називається files, зберігаються імена файлів, а також їх метадані, наприклад, розмір. А в іншій колекції, яка називається chunks, у вигляді невеликих сегментів зберігаються дані файлів, зазвичай сегментами по 256 кб.

Отже, для такої бази даних маємо колекції: familyBudget, transactions, users.

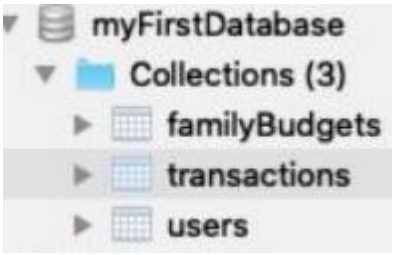
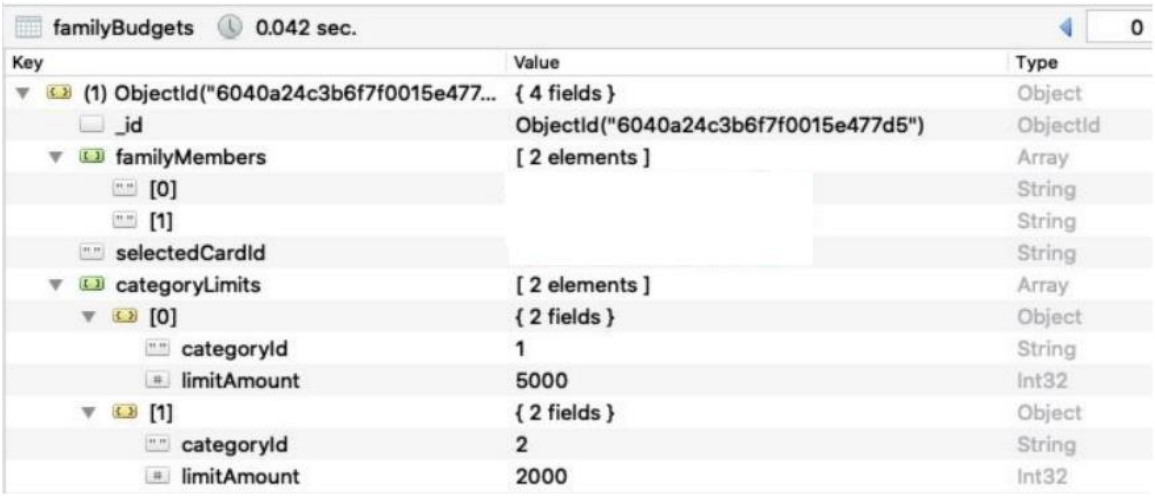


Рис. 2.3. Структура бази даних



familyBudgets 0.042 sec. 0

Key	Value	Type
(1) ObjectId("6040a24c3b6f7f0015e477...	{ 4 fields }	Object
_id	ObjectId("6040a24c3b6f7f0015e477d5")	ObjectId
familyMembers	[2 elements]	Array
[0]		String
[1]		String
selectedCardId		String
categoryLimits	[2 elements]	Array
[0]	{ 2 fields }	Object
categoryId	1	String
limitAmount	5000	Int32
[1]	{ 2 fields }	Object
categoryId	2	String
limitAmount	2000	Int32

Рис. 2.4. Колекція familyBudget

В колекції сімейного бюджету бачимо членів родини, обрану карта та встановлені ліміти на даний рахунок.

Key	Value	Type
(1) ObjectId("6040a2843b6f7f0015e47...	{ 6 fields }	Object
_id	ObjectId("6040a2843b6f7f0015e477d6")	ObjectId
type	StatementItem	String
data	{ 2 fields }	Object
account		String
statementItem	{ 12 fields }	Object
id	dQ7q8bEsy4zihSJV	String
time	1614848635	Int32
description	Happy Paw	String
mcc	4900	Int32
amount	-98	Int32
operationAmount	-98	Int32
currencyCode	980	Int32
commissionRate	0	Int32
cashbackAmount	0	Int32
balance	33200	Int32
hold	true	Boolean
receiptId	EE53-2T94-T74P-TTHK	String
familyBudgetId	6040a2	String
bankName	monobank	String
ownerId	6040a2263b6	String
(2) ObjectId("6040d2033b6f7f0015e4...	{ 5 fields }	Object
(3) ObjectId("6040d73a3b6f7f0015e4...	{ 5 fields }	Object
(4) ObjectId("6040ef7a3b6f7f0015e47...	{ 5 fields }	Object
(5) ObjectId("604117043b6f7f0015e47...	{ 6 fields }	Object

Рис 2.5. Колекція transactions

В колекції транзакцій бачимо кожну транзакцію, яка має: ідентифікатор, тип, аккаунт користувача (що її здійснив), час здійснення, опис, MCC код, суму, код валюти, комісію, кешбек, баланс рахунку, ідентифікатор чеку, ідентифікатор бюджету, ім’я банку та ідентифікатор власника.

Рис. 2.6. Колекція users

В колекції користувачів маємо ідентифікатор кристувача, ідентифікатор чату в Telegram, ім’я користувача, мовний код та токен Monobank.

2.5. Контроль змін та версій за допомогою GitHub

В основі GitHub лежить Git, проект з відкритим кодом, започаткований творцем Linux Лінусом Торвальдсом. Метью Маккаллоу, тренер з GitHub, пояснює, що Git, як і інші системи контролю версій, керує та зберігає редакції проектів. Хоча він в основному використовується для коду, Маккаллоу каже, що Git можна використовувати для управління файлами будь-якого іншого типу, наприклад документами Word або проектами Final Cut. Подумайте про це як про систему подання кожного проекту документа.

Деякі попередники Git, такі як CVS та Subversion, мають центральне "сховище" всіх файлів, пов'язаних з проектом. Маккалоу пояснює, що коли розробник вносить

зміни, ці зміни вносяться безпосередньо до центрального сховища. За допомогою розподілених систем контролю версій, таких як Git, якщо ви хочете внести зміни до проекту, ви копіюєте ціле сховище у власну систему. Ви вносите зміни в локальну копію, а потім «реєструєте» зміни на центральному сервері.

МакКаллоу каже, що це заохочує надання більш детальних змін, оскільки вам не потрібно підключатися до сервера щоразу, коли ви вносите зміни. GitHub - це сервіс хостингу сховищ Git, але він додає багато власних функцій. Хоча Git - це інструмент командного рядка, GitHub надає графічний інтерфейс на основі Інтернету. Він також забезпечує контроль доступу та кілька функцій співпраці, таких як вікі та основні інструменти управління завданнями для кожного проекту. Флагманською функціональністю GitHub є «розгалуження» - копіювання сховища з облікового запису одного користувача в інший.

Це дозволяє взяти проект, до якого ви не маєте права писати, та змінити його під своїм обліковим записом. Якщо ви внесли зміни, якими хочете поділитися, ви можете надіслати сповіщення, яке називається "запит на витяг", оригінальному власнику. Потім цей користувач може, натиснувши кнопку, об’єднати зміни, знайдені у вашому репо, з оригінальним репо.

Ці три функції - fork, pull request і merge - саме це робить GitHub настільки потужним. Грегг Поллак із Code School (який щойно запустив клас під назвою TryGit) пояснює, що до GitHub, якщо ви хотіли взяти участь у проекті з відкритим кодом, вам потрібно було вручну завантажити вихідний код проекту, внести зміни локально, створити список змін, що називаються “патч”, а потім надішліть патч електронною поштою до супровідника проекту. Тоді супровідник повинен був би оцінити цей патч, можливо, надісланий зовсім незнайомим особою, і вирішити, чи об’єднувати зміни. Тут мережевий ефект починає відігравати роль у GitHub, пояснює Поллак. Коли ви подаєте запит на вилучення, супровідник проекту може бачити ваш профіль, який включає всі ваші внески на GitHub.

Якщо ваш патч прийнятий, ви отримуєте запис до авторів на оригінальному сайті, і він відображається у вашому профілі. Це як резюме, яке допомагає

супровіднику визначити вашу репутацію. Чим більше людей та проектів на GitHub, тим краща ідея може бути у розробника проектів потенційних авторів.

Виправлення також можна публічно обговорювати. Навіть для тих, хто обслуговує користувачів, які в кінцевому підсумку не використовують інтерфейс GitHub, GitHub може спростити управління внесками. "У підсумку я все одно просто завантажую патч або зливаю його з командного рядка, а не з кнопки злиття", - каже Ісаак Шлютер, супровідник платформи розробки з відкритим кодом Node.js. "Але GitHub забезпечує централізоване місце, де люди можуть обговорювати патч".

Зниження бар'єру для вступу демократизує розробку з відкритим кодом та допомагає молодим проектам рости. "Без GitHub Node.js не був би таким, яким є сьогодні", - говорить Шлютер. Окрім своїх відкритих сховищ із відкритим кодом, GitHub також продає приватні сховища та локальні екземпляри свого програмного забезпечення для підприємств. Ці рішення, очевидно, не можуть повною мірою скористатися мережевим ефектом GitHub, але вони можуть скористатися перевагами функцій співпраці. Ось як GitHub заробляє гроші, але на цьому ринку він не єдиний.

Atlassian придбав конкурента під назвою BitBucket у 2010 році. На початку цього року Atlassian запустив Stash, продукт, що дозволяє розміщувати приватні локальні сховища Git з функціями співпраці у стилі BitBucket / GitHub. Компанія також продає інструменти для співпраці розробників, такі як відстежувач помилок Jira та wiki Confluence. Конкуренція від Atlassian, яка взяла 60 мільйонів доларів на фінансування від Accel Partners у 2010 році, може допомогти пояснити, чому GitHub взяла цей раунд фінансування, та натякнути на деякі можливі подальші напрямки для компанії. Наприклад, Шлютер каже, що функція відстеження проблем від GitHub може з часом скласти конкуренцію JIRA для деяких проектів. Гроші можуть бути в приватному та локальному хостингу, але любов - у державних сховищах.

Мабуть, найголовніше те, що GitHub став Олександрійською бібліотекою для прикладів коду. Оскільки Git заохочує детально реєструвати зміни, програмісти, будь то абсолютні новачки чи експерти, можуть простежити кроки деяких найбільших розробників у світі та з'ясувати, як вони вирішували неприємні проблеми.

РОЗДІЛ 3
АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

3.1 Структура проекту

Почнемо з огляду файлової структури.

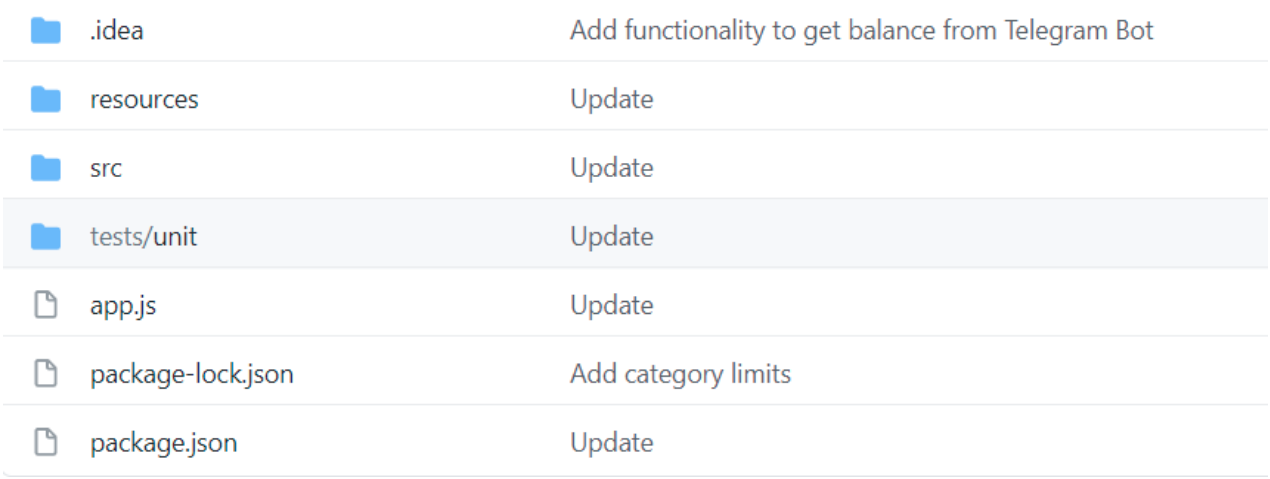


Рис.3.1. Файлова структура

У папці .idea зберігається інформація про налаштування середовища, яке використовувалося для написання коду – Intellij Idea. Вона не є обов'язковою і не відноситься до роботи чат-боту, проте є важливою при роботі з різних комп'ютерів.

Наповнення файлу app.js:

```
require('dotenv').config();

const path = require('path');
const { I18n } = require('i18n');
const useServer = require('./src/server');
```

```
onst useTelegramBot = require("./src/telegramBot");
const useDBGateway = require("./src/dbGateway");
const useAPIGateway = require("./src/apiGateway");
const useBotGateway = require("./src/botGateway");

if (!process.env.PORT) {
    throw TypeError('PORT variable is required')
}

if (!process.env.TELEGRAM_BOT_TOKEN) {
    throw TypeError('TELEGRAM_BOT_TOKEN variable is required')
}

if (!process.env.BASE_URL) {
    throw TypeError('BASE_URL variable is required')
}

if (!process.env.MONGO_URI) {
    throw TypeError('MONGO_URI variable is required')
}

if (!process.env.MONGO_DB_NAME) {
    throw TypeError('MONGO_DB_NAME variable is required')
}

(async () => {
    const i18n = new I18n({
        locales: ['en', 'ru', 'ua'],
        directory: path.join(__dirname, 'resources', 'locales')
    });
```

```
    const dbGateway = await useDBGateway();
    const apiGateway = await useAPIGateway();
    const telegramBot = await useTelegramBot({dbGateway, apiGateway, i18n});
    const botGateway = await useBotGateway({telegramBot});

    useServer({
        telegramBot, dbGateway, botGateway, i18n, apiGateway, });
})();
```

Отже, даний документ є відправною точкою для всього проекту: запускаються модулі, сервер та перевіряються змінні оточення.

Далі – package.json:

```
{
  "name": "fbos",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node app.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/yegorHeiz/fbos.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/yegorHeiz/fbos/issues"
  },
  "homepage": "https://github.com/yegorHeiz/fbos#readme",
  "description": "",
  "dependencies": {
```

```
    "axios": "^0.21.0",
    "body-parser": "^1.19.0",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "i18n": "^0.13.2",
    "joi": "^17.3.0",
    "lodash": "^4.17.20",
    "moment": "^2.29.1",
    "mongodb": "^3.6.3",
    "node-telegram-bot-api": "^0.50.0",
    "numeral": "^2.0.6"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "mocha": "^8.2.1"
  }
}
```

Тут можна побачити усі бібліотеки що використовуються, та залежності для розробки (devDependencies), які не попадають на продакшен.

У папці tests/unit зберігаються тести для боту, що покривають основну технічну частину, частина функцій тестувалась користувачем.

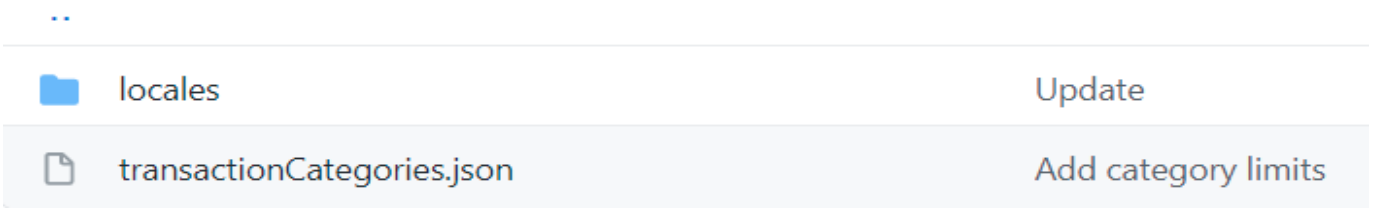


Рис. 3.1. Наповнення папки resources

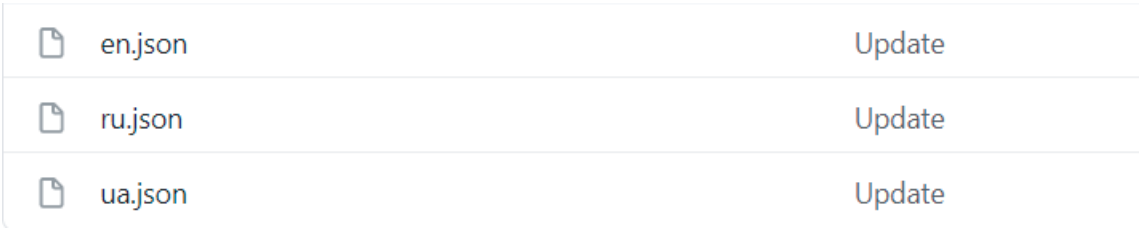


Рис. 3.2. Наповнення папки locales

Папка resources зберігає файли локалізаціїї бота на трьох мовах (рис. 3.3) та файл з класифікацією MCC-кодів по категоріям витрат:

```
"cafeAndRestaurants": {
   "id": "2",
   "title": "cafeAndRestaurants",
   "mccCodes": [
     5811,
     5812,
     5813,
     5814
   ]
 },
 "cinema": {
   "id": "3",
   "title": "cinema",
   "mccCodes": [
     7829,
     7832,
     7841
   ]
 },
```

Приклад локалізаціїї:

```
"cafeAndRestaurantsShort": "Кафе і ресторани",
```

"cinema": "Кіно. Послуги та товари кінотеатрів, оренда і купівля товарів в спеціалізованих магазинах",

"cinemaShort": "Кіно",

"taxi": "Таксі. послуги таксі",

"taxiShort": "Таксі",

"medicine": "Краса і медицина. Товари та послуги в масажних або косметичних салонах, і аптеки",

"medicineShort": "Краса і медицина",

Переходимо до папки src, де знаходиться основний код.

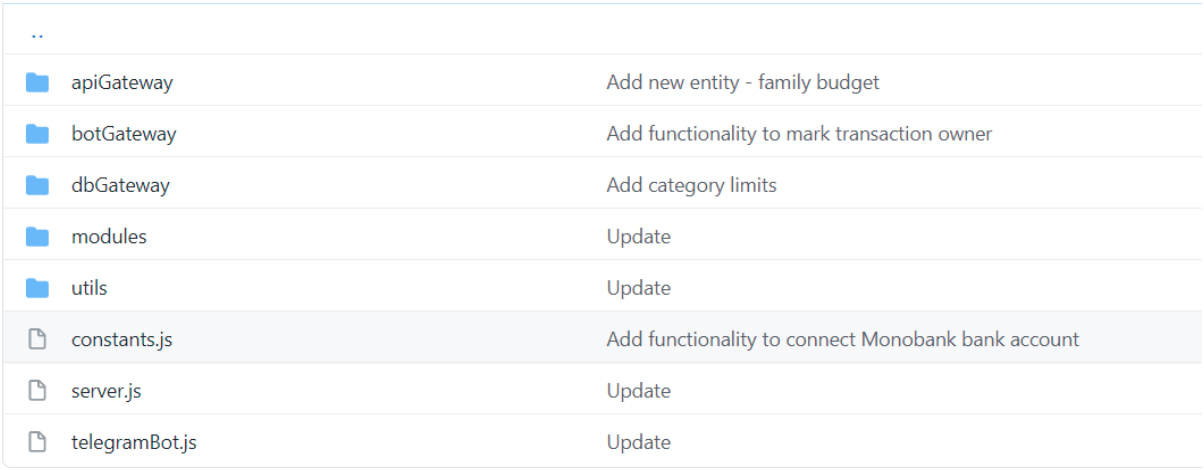


Рис. 3.3. Наповнення папки src

У теці apiGateway знаходиться модуль через який проходять усі запити на Monobank. Тут маємо запити на данні користувача та встановлення веб-хуку для того щоб Monobank надавав нашому серверу інформацію, що надходить:

```
const axios = require('axios');

const MONOBANK_API_BASE_URL = 'https://api.monobank.ua';

module.exports = () => {
    return {
        users: {
```

```
            getUserInfo: async bankToken => {
                const url = `${MONOBANK_API_BASE_URL}/personal/client-info`;

                const response = await axios.get(url, {headers: {'X-Token': bankToken}});
                return response.data;
            },
            setupWebhook: async (bankToken, bankName, familyBudgetId) => {
                const url = `${MONOBANK_API_BASE_URL}/personal/webhook`;
                const webHookUrl = `${process.env.BASE_URL}/webhooks/familyBudgets/${familyBudgetId}/banks/${bankName}`;

                const response = await axios.post(url, {webHookUrl}, {headers: {'X-Token': bankToken}});
                return response.data;
            }
        }
    };
```

Далі маємо botGateway, що призначений для того, щоб зв’язуватися з чат-ботом – надсилати повідомлення у потрібний чат:

```
const {TELEGRAM} = require('./../constants');

module.exports = ({telegramBot}) => {
    const sendMessage = (chatID, text, options = {}) => {
        return telegramBot.sendMessage(chatID, text, options)
    };

    return {
        [TELEGRAM]: {
```

```
        sendMessage        }
    }};
```

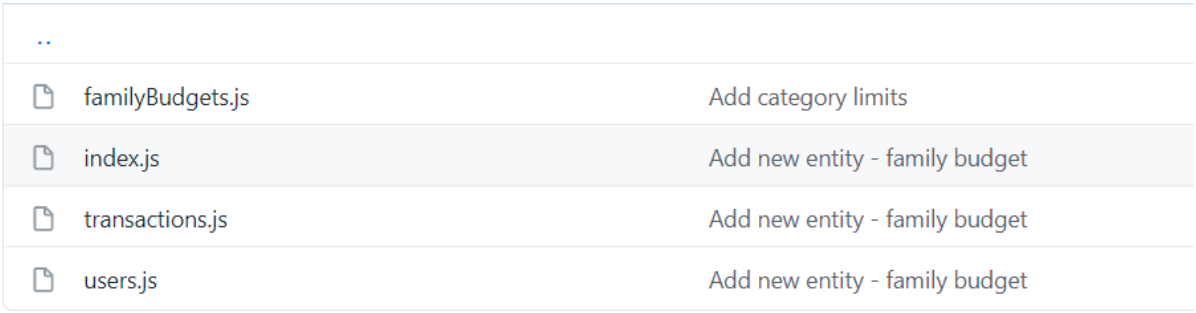


Рис. 3.4. Наповнення папки dbGateway

Початковим файлом для модуля бази даних є index.js, де завантажуються та експортуються користувачі, транзакції та дані про сімейний бюджет:

```
const {MongoClient} = require('mongodb');

const useUsers = require('./users');
const useTransactions = require('./transactions');
const useFamilyBudgets = require('./familyBudgets');

module.exports = async () => {
    const client = await MongoClient.connect(process.env.MONGO_URI);
    const db = client.db(process.env.MONGO_DB);

    return {
        users: useUsers({db}),
        transactions: useTransactions({db}),
        familyBudgets: useFamilyBudgets({db}),
    }
};
```

У файлі familyBudgets.js маємо фунцію оновлення колекції familyBudgets або створення у ній нового запису і встановлення ліміту. Також можна отримати ідентифікатор власника картки або члена родини:

```
const _ = require('lodash');
const {ObjectID} = require('mongodb');
module.exports = ({db}) => {
  const familyBudgetsCol = db.collection('familyBudgets');

  const upsertFamilyBudget = async (familyMemberId, selectedCardId) => {
    const query = {
      $or: [
        {selectedCardId},
        {familyMembers: familyMemberId}
      ]
    };

    const existingFamilyBudget = await familyBudgetsCol.findOne(query);

    if (existingFamilyBudget) {
      await familyBudgetsCol.updateOne(
        query,
        {
          $set: {
            familyMembers:  _.uniq([...existingFamilyBudget.familyMembers, familyMemberId]),
            selectedCardId
          }
        }
      );
```

```
            return familyBudgetsCol.findOne(query);
        }

        const {insertedId} = familyBudgetsCol.insertOne({
            familyMembers: [familyMemberId],
            selectedCardId
        });

        return familyBudgetsCol.findOne({_id: insertedId})
    };

    const getById = id => {
        return familyBudgetsCol.findOne({_id: ObjectID(id)})
    };

    const getByFamilyMemberId = familyMemberId => {
        return familyBudgetsCol.findOne({familyMembers: familyMemberId})
    };

    const setCategoryLimit = async (id, categoryId, limitAmount) => {
        const familyBudgetBeforeUpdate = await familyBudgetsCol.findOne({_id:
ObjectID(id)});

        await familyBudgetsCol.updateOne({_id: ObjectID(id)}, {
            $set: {
                categoryLimits: [
                    ...(familyBudgetBeforeUpdate.categoryLimits || []).filter(x =>
x.categoryId !== categoryId),
                    {categoryId, limitAmount}
                ]
```

```
        }
    });

    return familyBudgetsCol.findOne({_id: ObjectID(id)});
};
return {upsertFamilyBudget, getById, getByFamilyMemberId, setCategoryLimit};
};
```

Те ж саме є для транзакцій (створення, оновлення, отримання транзакції по ідентифікатору сім'ї) та для користувачів.

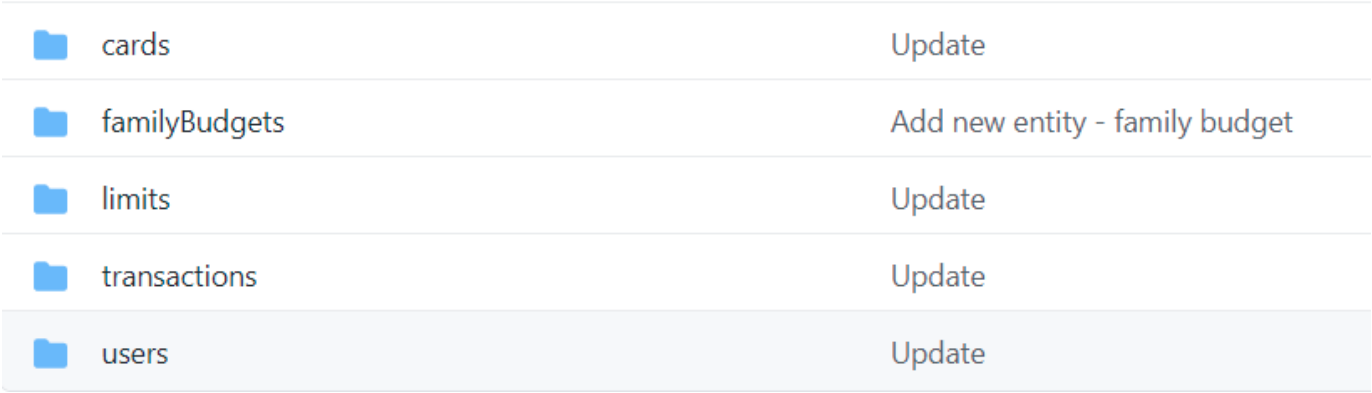


Рис. 3.5. Наповнення теки modules

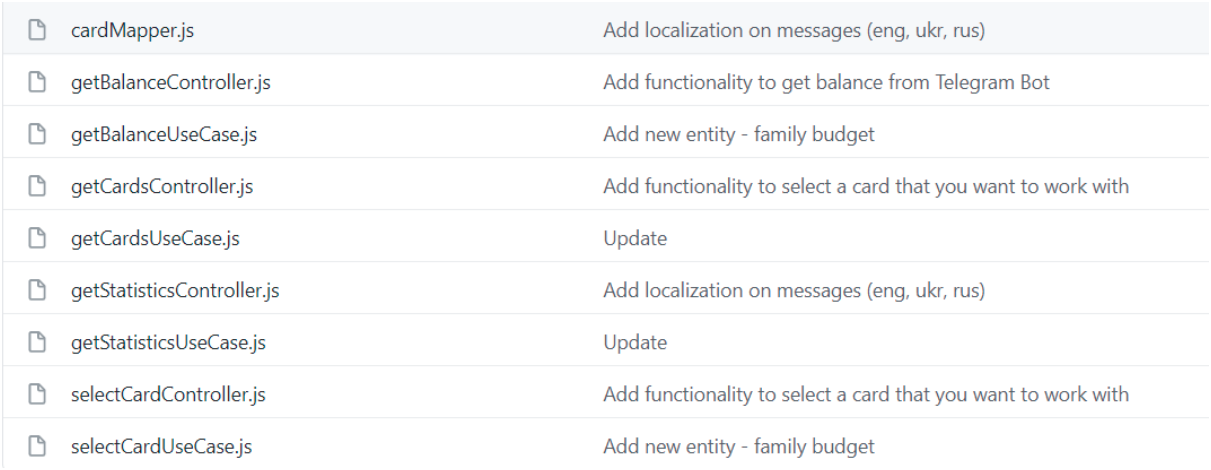


Рис. 3.6. Наповнення теки cards

У теці cards файл cardMapper.js об'єкт карти перетворює на рядок. Файл getBalanceUseCase.js знаходить користувача, його сім'ю, інформацію про

користувача, знаходить карту яка обрана для ведення бюджету і надсилає номер карти в Telegram.

Файл getCardsUseCase знаходить користувача, його інформацію і локалізує інформацію для підключення з ботом.

Файл getStatisticsUseCase.js повертає статистику користувачеві в чат.

Файл selectCardUseCase.js надсилає повідомлення про те, що карта обрана та оновлює цю інформацію в базі даних.

Усі файли з закінченням “Controller” визивають однойменні функції.

Модуль familyBudget (рис.3.5) вносить нові данні або оновлює їх у колекції.

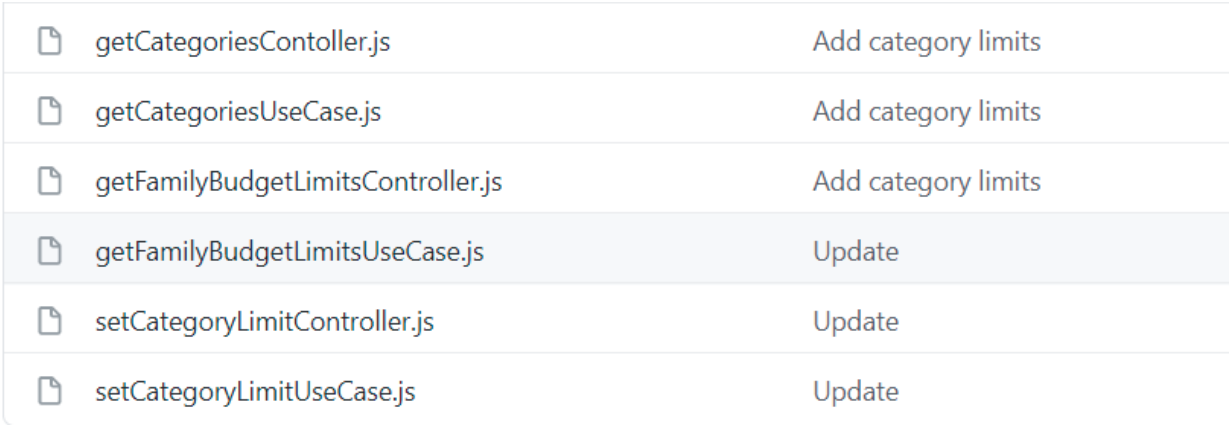


Рис. 3.7. Наповнення теки limits

Модуль limits використовується для встановлення так контролю лімітів на обрані категорії.

Файл getCategoriesUseCase.js слугує для розпізнання назв категорій та для передачі даних в Telegram.

Файл getFamilyBudgetLimitsUseCase.js повертає користувачеві усі його існуючі ліміти в чат.

Файл getCategoryLimitUseCase.js використовується для встановлення лімітів на обрану категорію.

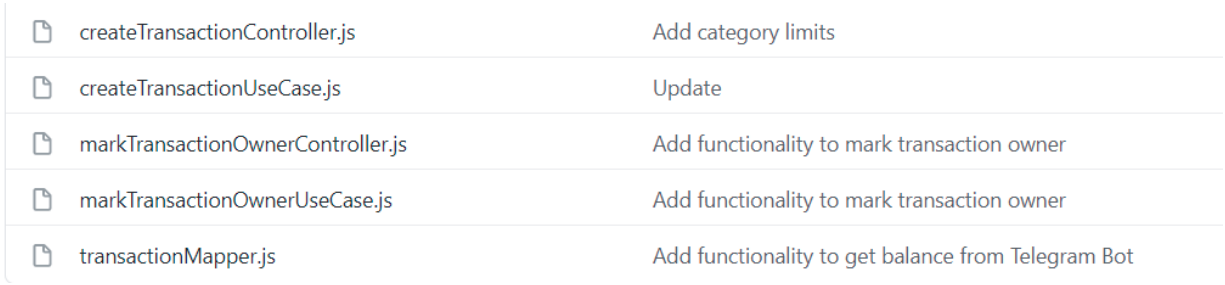


Рис. 3.8. Наповнення теки transactions

Файл createTransactionUseCase.js слугує для створення транзакції та передачі даних про неї в чат з користувачем.

Файл markTransactionOwnerUseCase.js використовується для визначення того, хто її провів – один із членв родини або разом.

Файл transactionMapper.js робить з об’єкта транзакції рядкове значення.

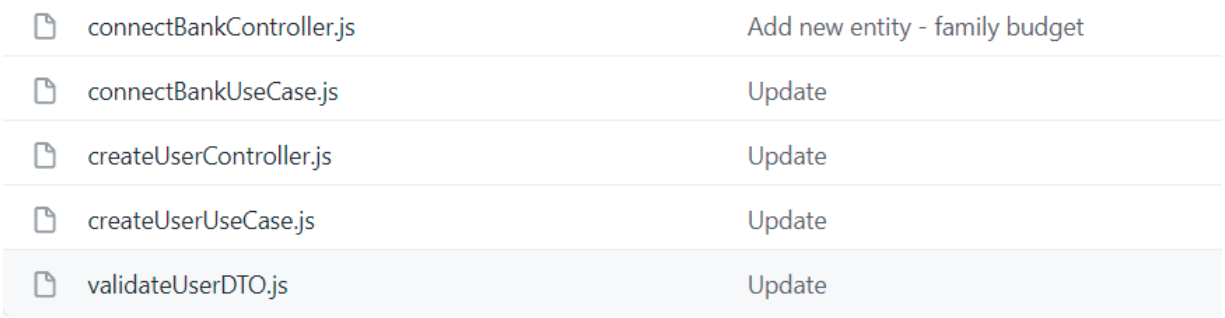


Рис. 3.9. Наповнення теки users

Модуль users використовується для під’єднання до банку (connectBankUseCase.js) та створення запису користувача (createUserUseCase.js). Файл validateUserDTO.js слугує для передачі об’єкту між модулями та для його валідації:

```
const Joi = require('joi');

const schema = Joi.object({
    telegramChatId: Joi.number().required(),
    name: Joi.string().required(),
    languageCode: Joi.string().required()
});
```

```
module.exports = data => {
    const {error, value} = schema.validate(data);

    if (error) {
        throw error;
    } else {
        return value;
    }};
```

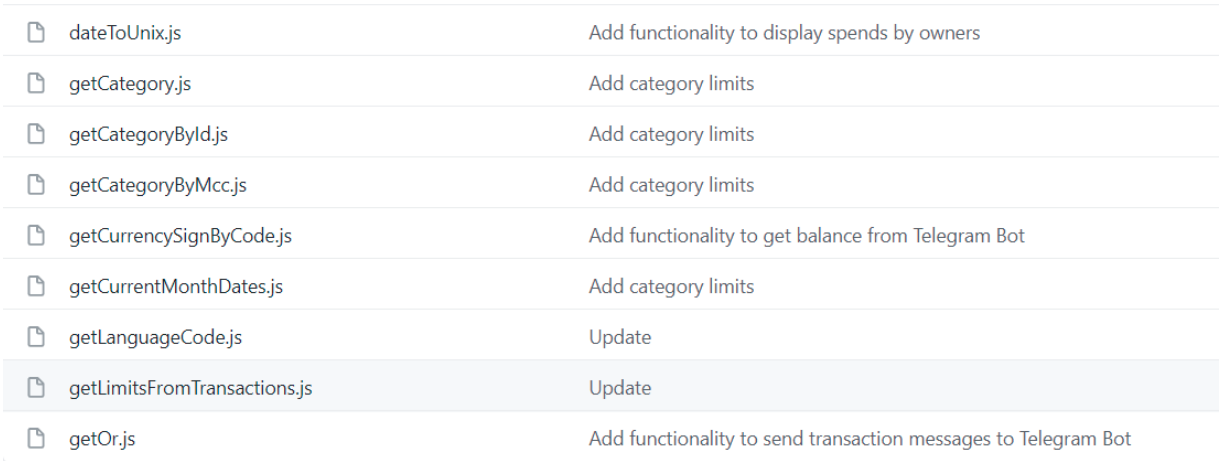


Рис. 3.10. Наповнення теки utils

У теці utils зібрані корисні функції – ініціалізація категорій по різним даним, ініціалізація валюти, мови, поточного місяця, лімітів із здійсненої транзакції.

3.2. Тестування

Перегляд роботи розробленого проекту:

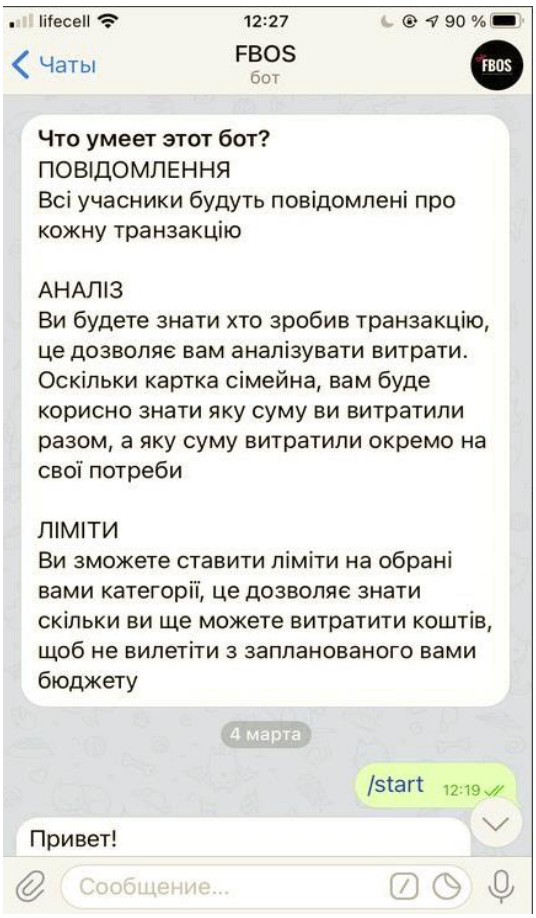


Рис. 3.11. Початкове повідомлення

Вперше відкривши FBOS чат-бот, ви отримаєте коротке представлення про основні функції та переваги даного бота.

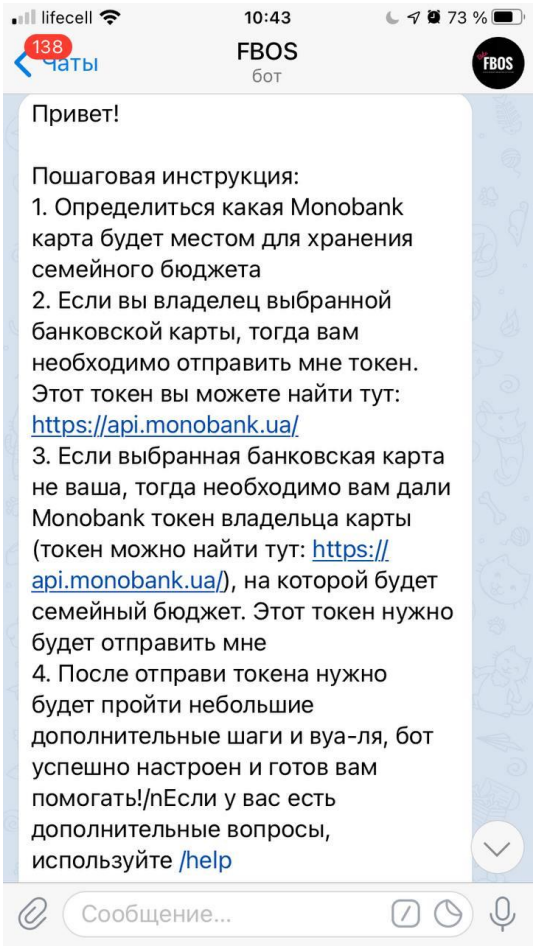


Рис.3.12. Інструкція по підключенню

Користувач зможе з легкістю під’єднати свій рахунок, адже бот надає чітку та просту інструкцію з необхідними посиланнями.

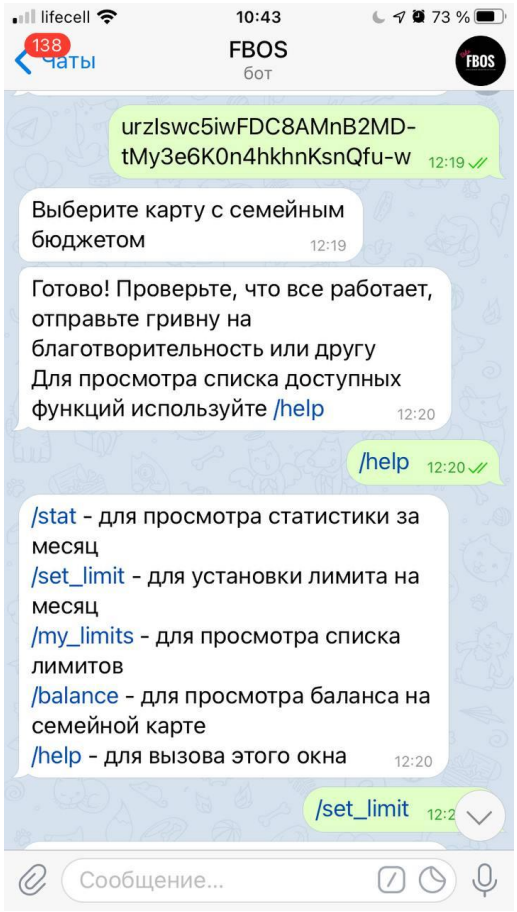


Рис.3.13. Підключення карти

Після отримання токену бот надає користувачеві список наявних карт, з яких можна обрати одну для ведення бюджету. Потім можливо ознайомитися зі всіма командами (тобто функціями), а саме – перегляд статистики за місяць, встановлення лімітів на категорію, перегляд списку встановлених лімітів, перегляд поточного балансу рахунку та виклик цього вікна.

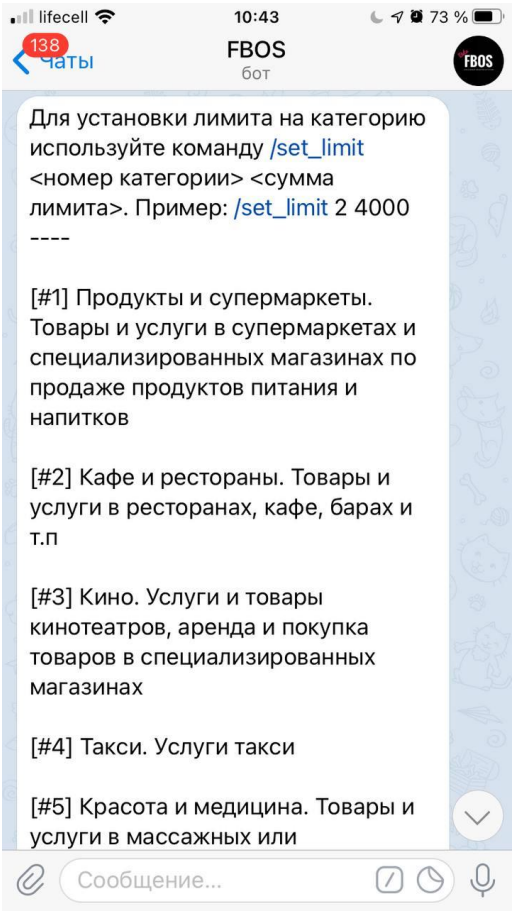


Рис.3.14. Встановлення лімітів

Після активації команди /set_limit ми бачимо таке вікно з повідомленням, де можна переглянути усі доступні категорії та інструкцію зі встановлення лімітів.

Рис.3.15. Перегляд лімітів та вигляд транзакцій

Доступна команда перегляду встановлених лімітів та визначення того, хто зробив транзакцію. Користувач може самостійно вказати хто витратив кошти.

Рис.3.16. Перегляд лімітів та вигляд транзакцій

На рис.3.16 можна ознайомитися з тим, як виглядає статистика за місяць та команда перевірки балансу.

ВИСНОВКИ

В результаті виконання дипломного проекту розроблено чат-бот для організації фінансів, що використовується для реальних потреб.

Спроектовано та розроблено програмний додаток, що передбачає:

- встановлення лімітів по категоріям,
- розподіл витрат,
- сповіщення про кожну транзакцію,
- автоматизація внесення інформації про витрати,
- аналіз витрат за вказаний період,
- мобільність версії.

Проаналізовано такі засоби розробки як Telegram API, Monobank API, GitHub, MongoDB.

Порівняно додатки для фінансового обліку.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bruce Eckel. Thinkingin Java. Prentice Hall [Електронний ресурс]. – 2006. - №6.
2. Design Goalsof the Java TM Programming Language [Електронний ресурс]. – Режим доступу: http://java.sun.com/docs/books/tutorial/getStarted/intro/changemylife.html
3. Different Isn't Always Better, But Better's Always Different [Електронний ресурс]. – 2007.
4. Feigenbaum, Barry. SWT, Swingor AWT: Which is right for you? [Електронний ресурс]. – 2006.
5. Goslingand McGilton. The Java Language Environment [Електронний ресурс]. – 1996.
6. J. Gosling, B. Joy, G. Steele, G. Brachda. TheJavaLanguageSpecification [Електронний ресурс]. – 2013. - №2. - Режим доступу: http://www.computerworld.com.au/index.php/id;1422447371;pp;3;fp;4194304;fpid;1
7. The Java Language Specification/ James Gosling, Bill Joy, Guy Steele, Gilad Bracha; Addison-Wesley. – 2015. - №3.
8. Google win scrucial API ruling, Oracle's case decimated/ Joe Mullin; Ars Technica. – 2012.
9. Effective Java/ Joshua Bloch; Prentice Hall. – 2018. - №2.
10. The Java History Time line. Офіційний сайт www.java.com [Електронний ресурс].
11. Барри Берд. Программирование на Java для чайников com [Електронний ресурс]/ Барри Берд. - Видавн.-книготорг. компанія «Діалектика», 2013.